MEMORIA
PRACTICA 2 - AED2
Problema 9

JOSE RAMON HOYOS BARCELO

Nombre	Grp	e-mail
Angel Ruiz Fernandez	G2.2	a.ruizfernandez@um.es
Alberto Velasco Cotes	G2.2	a.velascocotes@um.es

Cuenta de Mooshak: G2_36

1. Lista de problemas resueltos

ENVIO	PROBLEMA
135	C_AR
55	K_Ba

2. Resolucion de problemas

2.1. Avance rapido

2.1.1. Pseudocodigo y explicación

2.1.2. Programación del algoritmo

2.1.3. Estudio teórico

2.1.4. Estudio experimetal

2.1.5. Contraste de estudio teorico y experimental

2.2. Backtracking

2.2.1. Pseudocódigo y explicación

Esquema de backtracking de optimización base, con variables para representar el peso repartido a cada equipo, y el par mas optimizado, usando la diferencia entre ellos como vao (valor optimo actual).

La solución se recorre usando un vector ‘sol‘ que por cada peso del vector dado indica a que equipo va, A o B.

Dentro del bucle, antes de generar un hermano se deshace el anterior ya que usamos las mismas variables acumulativas. El hermano se genera incrementando la combinación del vector solucion y aplicando la solución a los pesos de los equipos (cumulativo).

Si es solución (se usan todos los pesos y los equipos se diferencian en no más de 1), se compara con la solución optima anterior y se asigna si es más optimo. Si no es solución, se comprueba si esta rama puede dar soluciones, en cuyo caso se profundiza (busqueda primero en profundidad) y si no se descarta el subarbol.

Se puede profundizar si esta por debajo del nivel final, y los pesos restantes son mayores que la diferencia actual, y la diferencia en numero entre equipos es menor que los nieveles restantes.

Retroceder consiste en deshacer la sumación del peso escogido de las variables cumulativas, desdefinir la solucion en el nivel actual y volver al nivel anterior.

Concluido el bucle, asumiendo que se ha encontrado solución, se asegura de que el equipo A sea mas pequeño de B para resolver la indeterminación del orden, y se retorna la solución compuesta de los pesos de cada equipo.

```
Entero, Entero BT(pesos):
    Entero nivel, voa # valor optimo actual

    Entero pesoActA, pesoActB   # peso de equipos actual
    Entero pesoOptA, pesoOptB   # peso equipos optimo (solucion)

    Vector s(pesos.size)        # vector solución (cada peso 0 = A, 1 = B)

    mientras nivel >= 0:        # bucle iterativo
        # deshacer hermano anterior para generar otro
        si s[nivel] = 0:
            pesoActA = -pesoActA[nivel]
        sino s[nivel] = 1:
            pesoActB = -pesoActB[nivel]

        # generar hermano
        s[nivel]++
        # computar pesos sumando el peso actual al equipo correspondiente
        si s[nivel] = 0:
            pesoActA = pesoActA + pesos[nivel]
        sino si s[nivel] = 1:
            pesoActB = pesoActB + pesos[nivel]

        # diferencia entre equipos
        Entero diferencia = |pesoActA - pesoActB|

        # si es solución
        si nivel == n - 1:          # se usan todos los pesos
            si diferencia < voa:    # si es mas optimo, cambiar
                voa = diferencia
                pesoOptA = pesoActA
                pesoOptB = pesoActB

        # criterio de poda
        si
          nivel != n - 1
          && sum(pesos[nivel:]) > diff                          # poda 1300%
          && |contar(s, 0) - contar(s, 1)| < n - nivel:         # poda 114%
            nivel = nivel + 1
        # retroceder
        sino:
            mientras no s[nivel] < 1 and nivel >= 0:  # mientras queden herm.
                # deshacer
                si s[nivel] == 0:
                    pesoActA = pesoActA - pesos[nivel]
                sino si s[nivel] == 1:
                    pesoActB = pesoActB - pesos[nivel]
                s[nivel] = -1
                nivel = nivel - 1

    # solución, si son diferentes, el equipo A sea el mas ligero
    si pesoOptA > pesoOptB:
        intercambiar(pesoOptA, pesoOptB)

    retornar pesoOptA, pesoOptB
```

2.2.2. Programación del algoritmo

```
std::pair<int, int> bt(std::vector<int> pesos) {
    auto type = decltype(pesos)::value_type(0); // tipo a acumular
    int n = pesos.size();                       // tamaño de vectores
    // ordenar pesos: asignar pesos mas significativos primero
    std::sort(pesos.begin(), pesos.end(), std::greater<int>());

    int nivel = 0, voa = INT_MAX;   // nivel y valor optimo actual

    // por equipo
    int pesoActA = 0, pesoActB = 0; // variables cumulativas solucion actual
    int pesoOptA = 0, pesoOptB = 0; // variables optimas

    std::vector<int> s(n, -1);      // vector solucion

    while (nivel >= 0) {
        // deshacer hermano anterior y generar
        if (s[nivel] == 0)
            pesoActA -= pesos[nivel];
        s[nivel]++;
        if (s[nivel] == 0)
            pesoActA += pesos[nivel];   // sumar peso al equipo correspondiente
        else if (s[nivel] == 1)
            pesoActB += pesos[nivel];

        int diff = std::abs(pesoActA - pesoActB);   // computar diferencia
        // en solucion
        if ((nivel == n - 1) && (diff < voa)) {
            voa = diff;
            pesoOptA = pesoActA;
            pesoOptB = pesoActB;
        }
        // criterio de poda para profundizar
        if (
            (nivel != n - 1)
            && (std::accumulate(pesos.begin() + nivel, pesos.end(),type) > diff)
            && (std::abs(std::count(s.begin(), s.end(), 0)
                - std::count(s.begin(), s.end(), 1)) < n - nivel)
        )
            nivel++;
        // retroceder
        else
            while (!(s[nivel] < 1) && nivel >= 0) {
                // deshacer nivel
                if (s[nivel] == 0)
                    pesoActA -= pesos[nivel];
                else if (s[nivel] == 1)
                    pesoActB -= pesos[nivel];
                s[nivel] = -1;
                nivel--;
            }
    }

    // solucion ordenada
    if (pesoOptA > pesoOptB)
        std::swap(pesoOptA, pesoOptB);

    return std::pair<int, int>(pesoOptA, pesoOptB);
}
```

2.2.3. Estudio teórico

Cada peso puede estar en uno u otro equipo, indicando que la forma de backtracking es de arbol binario. El tamaño de los equipos solo puede diferir en una persona. Por tanto, mediante combinatoria habrá como maximo

$$s(n) = \binom{n}{n/2} + \binom{n}{n/2-1} + \binom{n}{n/2+1} \quad (1)$$

soluciones con n pesos. Esto se puede entender como que hay n C n/2 formas de escoger la mitad de pesos del vector pesos (que van a un equipo, y la otra mitad al otro equipo), más formas de legir n/2−1 y n/2+1 elementos.

Usando esta función y sabiendo que el arbol es binario, se puede inferir que va a tener

$$2^{\log_2(s(n))} - 2 \quad (2)$$

nodos, en el peor caso, sin contar el raiz. Como es normal en backtracking, este algoritmo es O(2^logn), bastante grande. Además en la implementación se ordena el vector de pesos tal que se asignen los pesos mas grandes primero, y los ajustes pequeños después, que tiene una complejidad de O(nlogn) en el caso promedio, pero comparado con la complejidad del backtracking, el tiempo de ordenación es despreciable. Así la ordenación permite la aplicación eficiente de las condiciones de poda, pero estos son muy dificiles de formular matematicamente. Se verán en el estudio experimental.

2.2.4. Estudio experimetal

2.2.5. Contraste de estudio teorico y experimental

3. Conclusiones